Introduction to Cocoa

Session 300

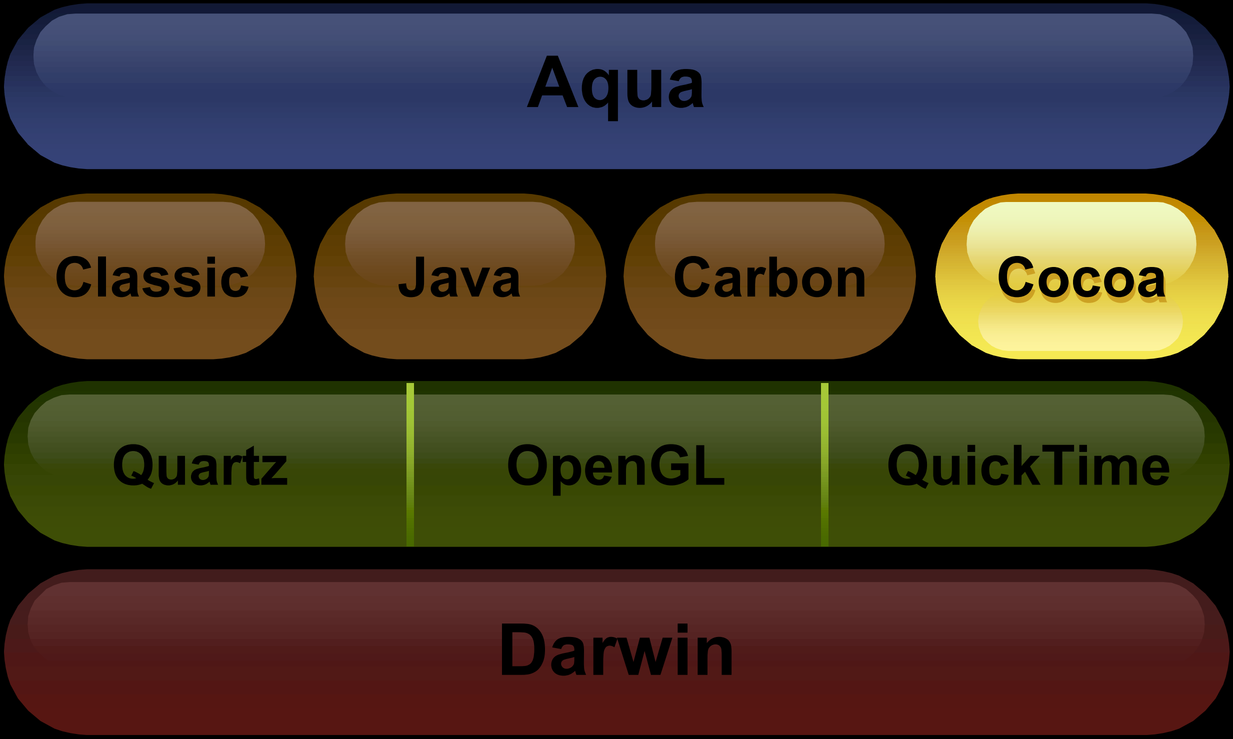
Introduction to Cocoa

Heather Hickman
Cocoa Evangelist

Introduction

- Cocoa is a powerful, full-featured, easy-to-use, object-oriented framework for Mac OS X development
- Rapid development and increased productivity
- Fastest way to full-featured applications for Mac OS X

Cocoa


Aqua
Classic
Java
Carbon
Cocoa
Quartz
OpenGL
QuickTime
Darwin

What You’ll Learn

- What makes Cocoa unique
- What is this language called Objective-C?
- A quick stroll through the class hierarchy and key concepts
- Resources for learning and becoming more productive using Cocoa

CodeWarrior v8.0

Matt Henderson
Technical Lead Mac OS Tools
Metrowerks

CodeWarrior and Cocoa

- Build Cocoa-based Software with CodeWarrior for Mac OS, v8.0
  - Metrowerks C/C++ compiler has supported Objective-C since 1998
  - CodeWarrior IDE 5.0 supports long file names
  - CodeWarrior IDE 5.0 integrates with Interface Builder

CodeWarrior and Cocoa Demonstration

Roadmap

003 Metrowerks Lunch Presentation	Hall 2 **Tues., 12:30pm**
CodeWarrior Birds of a Feather	Hall 2 **Tues., 7:30pm**
CodeWarrior Lounge Open all week	Hilton Plaza Room **8:00am–6:00pm**
Metrowerks Booth	Hall 1 **Mon., 5:00pm–8:00pm**

Project Builder

Mike Ferris
Manager, Project Builder Team

Project Builder—Cocoa Story

- Project Builder is made up of many “products”
  - The application
  - Frameworks
  - Plug-in bundles
  - Command-line tools, scripts
- Built by a total of 7 Project Builder projects
  - Layered and split along lines of functionality

The Main Projects That Build PB

- pbx_jamfiles: 5K lines of jam code
- toolsupport: 16K lines of Obj-C
- pbxbase: 70K lines of Obj-C (and a little C++)
- pbxide: 140K lines of Obj-C
- pbxdocviewer: 8K lines of Obj-C
- pbxprojectimporters: 3K lines of Obj-C

A Place to Stand

- NSWindowController
- NSDocument
- NSTextView / NSTextStorage
- NSTableView / NSOutlineView
- NSSplitView
- Key-value Coding / Property Lists

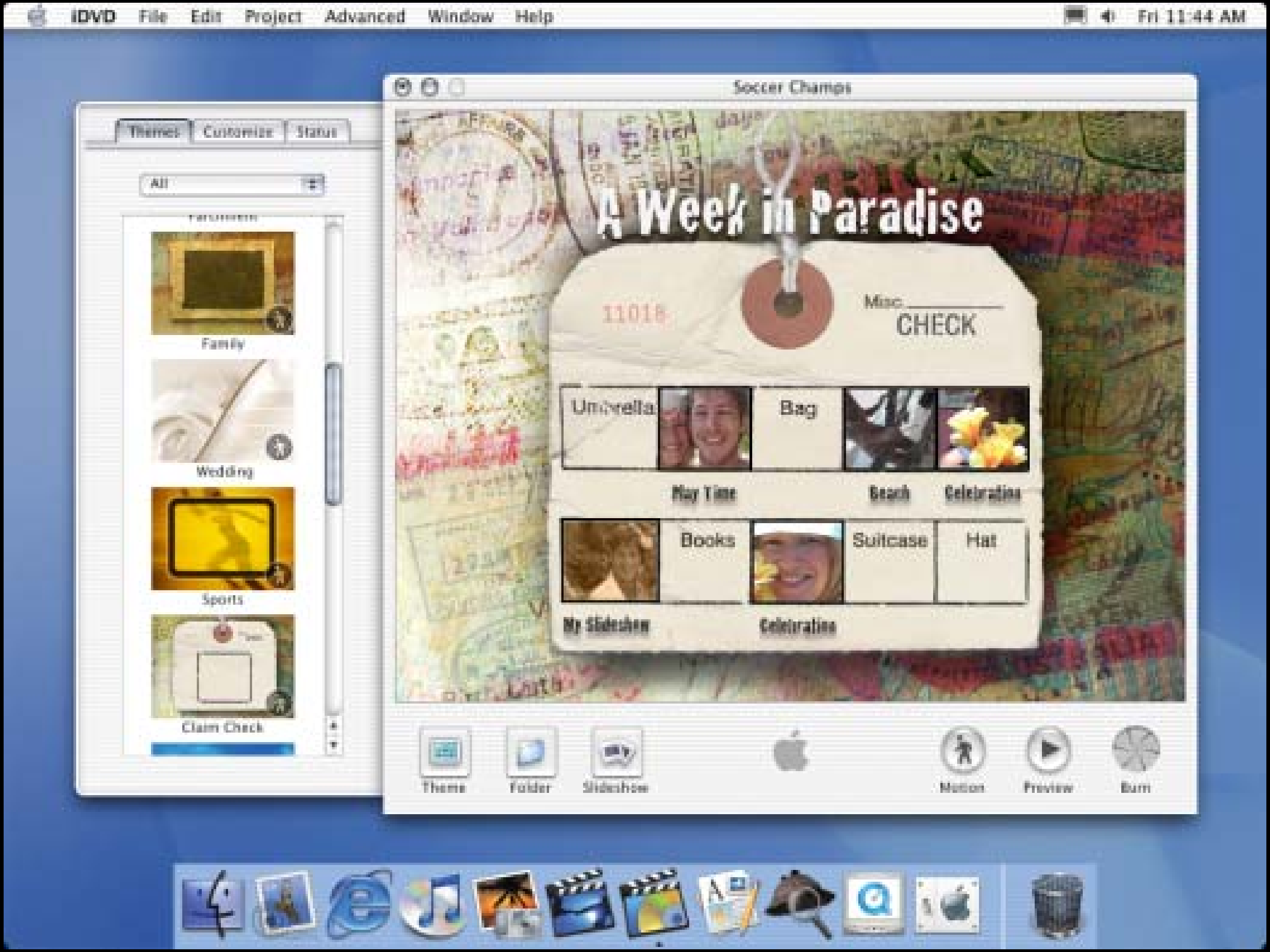
Roadmap

903 Exploring the Project Builder IDE	Hall 2 **Wed., 5:00pm**
908 Delivering With Project Builder	Hall 2 **Fri., 2:00pm**
902 AppleScript Studio Intro	Civic **Wed., 3:30pm**

iDVD2

Freddie Geier
Senior Director, New Media Engineering

Why Cocoa


iDVD File Edit Project Advanced Window Help
Fri 11:44 AM
Soccer Champs
Themes Customize Status
All
Family
Wedding
Sports
Claim Check
A Week in Paradise
11018
Misc.
CHECK
Umbrella
Bag
Play Time
Beach
Celebration
Books
Suitcase
Hat
My Slideshow
Celebration
Theme
Folder
Slideshow
Motion
Preview
Burn

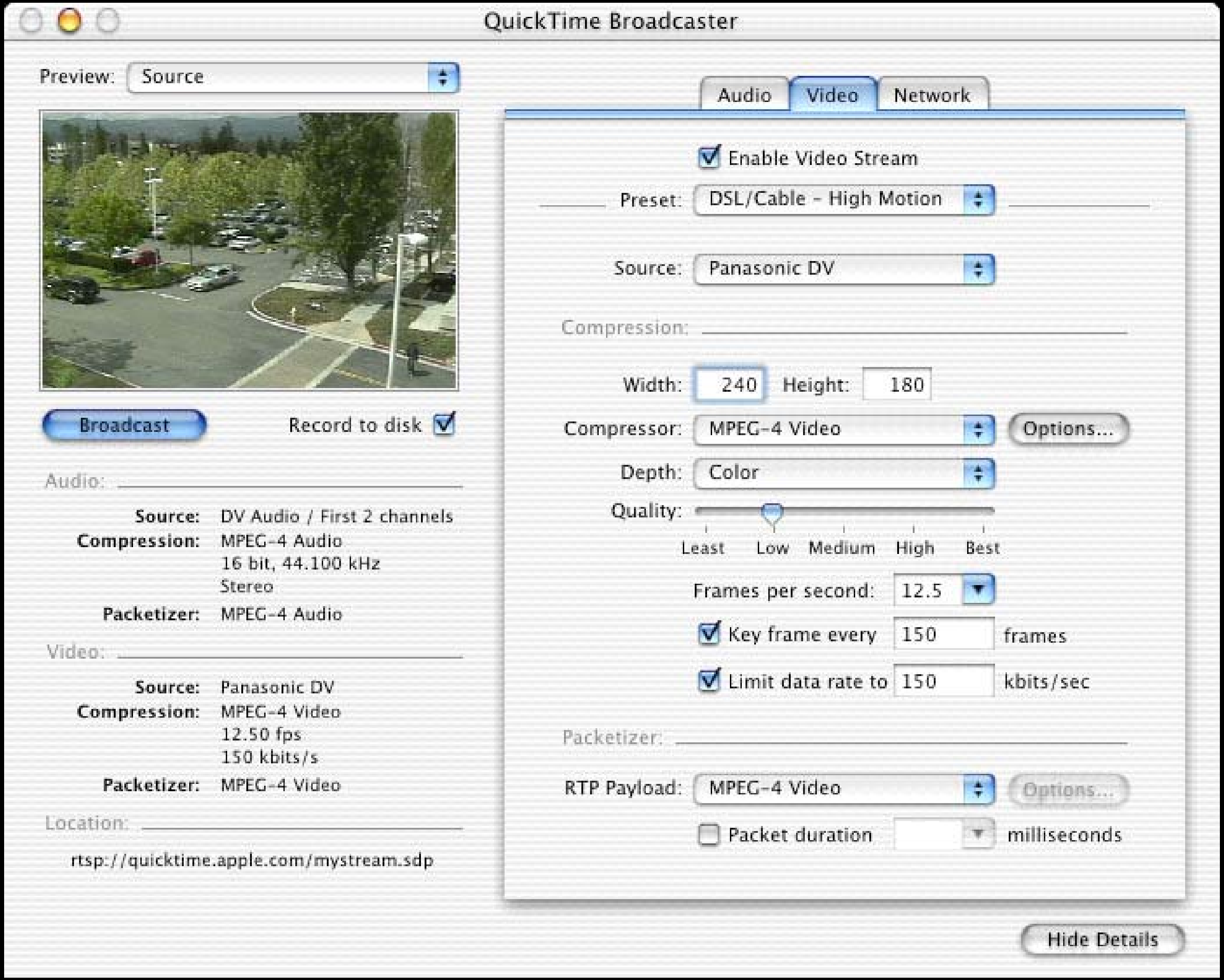
QuickTime

Adrian Baerlocher
QuickTime Engineering

QuickTime Broadcaster

- Cocoa application using QuickTime
- Classes encapsulate QuickTime functional groups
    - Sequence Grabber
    - Standard Compression
    - Broadcast APIs

QuickTime Broadcaster



Preview: Source

Record to disk ✓

Audio:

Source: DV Audio / First 2 channels
Compression: MPEG-4 Audio
16 bit, 44.100 kHz
Stereo
Packetizer: MPEG-4 Audio

Video:

Source: Panasonic DV
Compression: MPEG-4 Video
12.50 fps
150 kbits/s
Packetizer: MPEG-4 Video

Location:

rtsp://quicktime.apple.com/mystream.sdp

✓ Enable Video Stream

Preset: DSL/Cable - High Motion

Source: Panasonic DV

Compression:

Width: 240 Height: 180

Compressor: MPEG-4 Video

Depth: Color

Quality: Low

Frames per second: 12.5

✓ Key frame every 150 frames

✓ Limit data rate to 150 kbits/sec

Packetizer:

RTP Payload: MPEG-4 Video

☐ Packet duration milliseconds

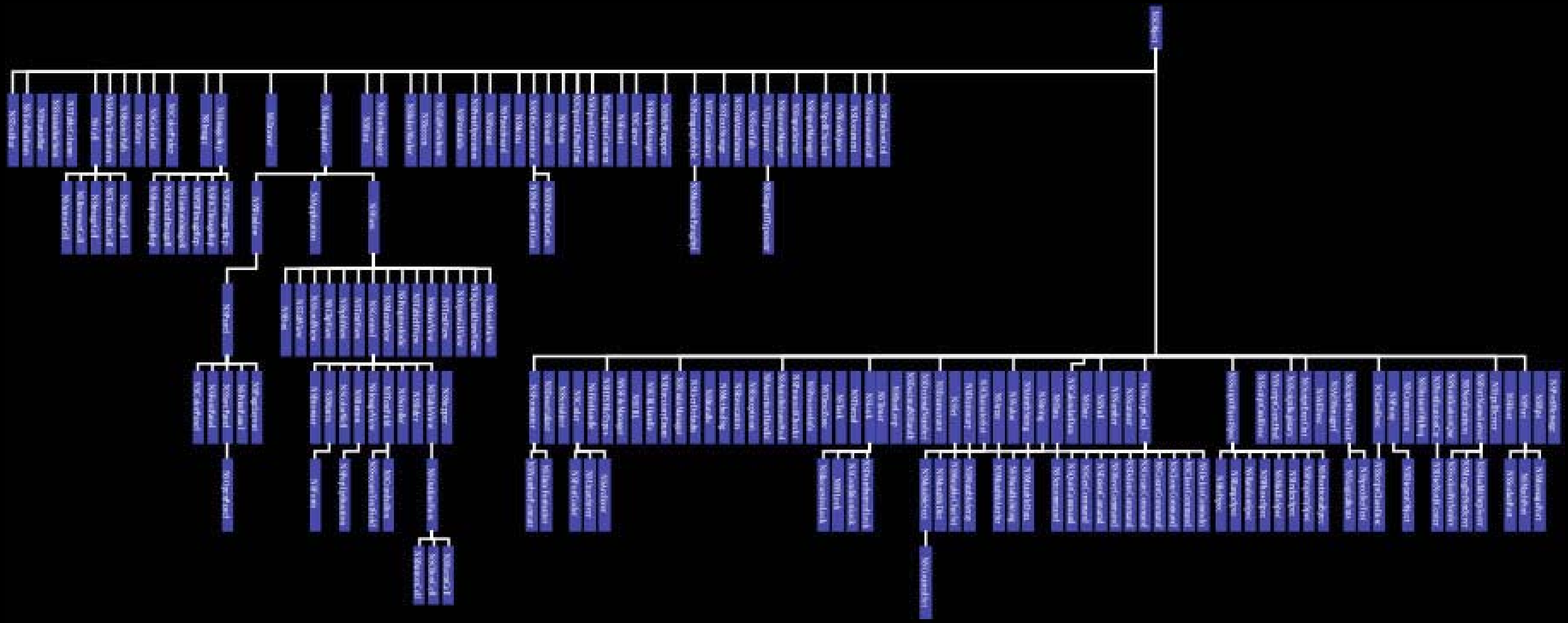
Roadmap

600 The State of QuickTime in 2002	Room A2 **Wed., 9:00am**
601 Building QuickTime-Savvy Apps	Room A2 **Wed., 10:30am**
Hands on With the QuickTime Engineers	Room G **Tues. thru Fri. 1:00pm–4:00pm**

Introduction to Cocoa

Matthew Formica
Cocoa DTS Engineer

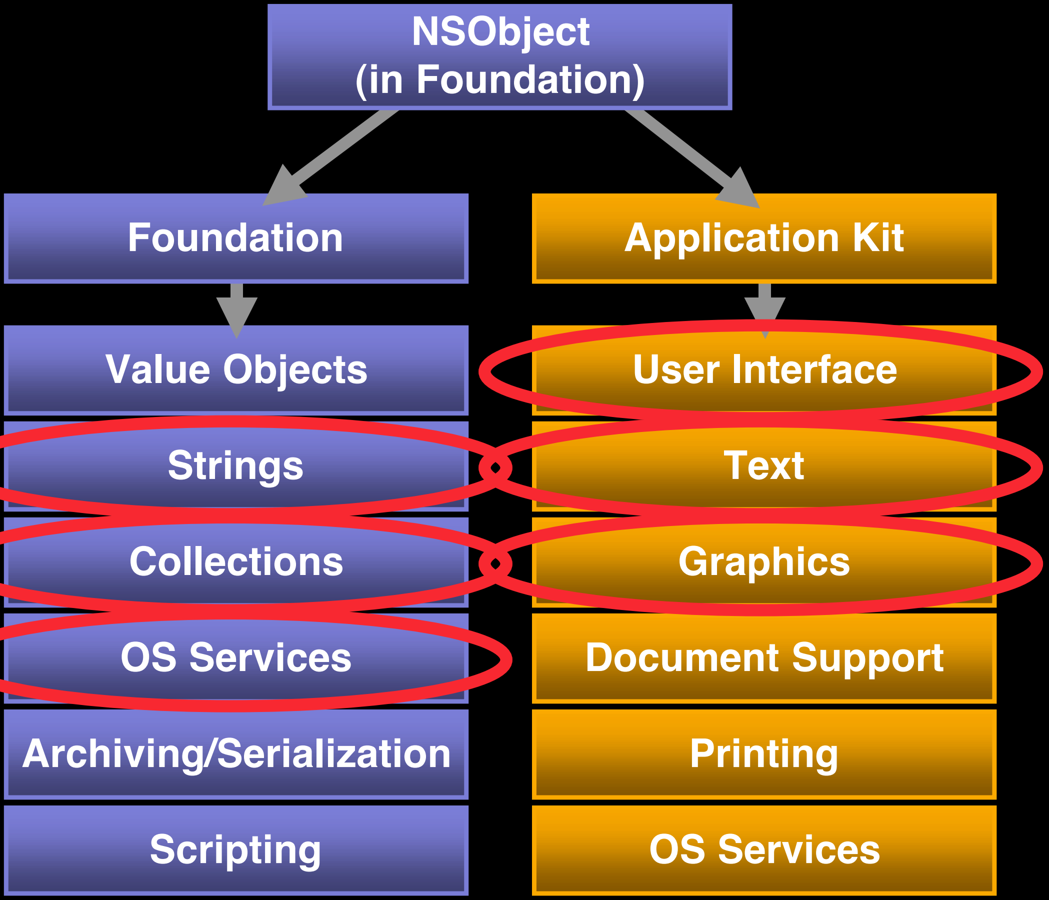
Full Featured



The Cocoa Hierarchy

Foundation

- Non-GUI
- Basic Data Structures
- Networking
- File System
- Scripting
- Accessibility
- And More!



AppKit

- GUI Widgets
- Text System
- Drawing
- Documents
- Printing
- And More!

Which Language to Use?

- Cocoa APIs are in Objective-C (ObjC) and Java
- You can plug your C and C++ code into ObjC as well
- Use the language that fits your needs
- We'll focus on ObjC today

Quick Overview of Objective-C

- Small superset of C
    - ANSI C
    - Some additional syntax
    - A few additional types
- Dynamic object runtime

Quick Overview of Objective-C

- Valid Objective-C (ObjC) code!

```
int i;
for (i=0; i<5; i++)
{
        printf(“Hello, World!\n”);
}
```

Weak Typing

- Objective-C determines dynamically at runtime what class a given object is!
- Generic “id” object type can refer to an object of any class
- Even typed objects can point to objects of other classes at compile-time/runtime (only warnings are produced)

Unique Messaging Syntax

For example…

$$\underbrace{\texttt{[myObject}}_{\textbf{object}}\ \underbrace{\texttt{doSomething]}}_{\textbf{message}}\texttt{;}$$

What is happening here?

- It’s a message being sent, not a function being called
- Messaging nil is a no-op
- Watch out for misspellings of method names—or you won’t be able to respond to the right messages!

Unique Messaging Syntax

For example… with named parameters

$$\underbrace{\texttt{[myObject}}_{\textbf{object}}\ \underbrace{\texttt{doWithThis:}}_{\textbf{message}}\ \underbrace{\texttt{otherObj];}}_{\textbf{parameter}}$$

$$\underbrace{\texttt{[myObject}}_{\textbf{object}}\ \underbrace{\texttt{doWithThis:}}_{\textbf{Msg-part1}}\ \underbrace{\texttt{object1}}_{\textbf{parm1}}\ \underbrace{\texttt{andThis:}}_{\textbf{Msg-part2}}\ \underbrace{\texttt{object2];}}_{\textbf{parm2}}$$

“**doWithThis:**” and “**doWithThis:andThis:**” are called *selectors*, identifying which message will be sent to a receiver

Class Definitions

```
#import <Cocoa/Cocoa.h> //like #include with #pragma once

//MyClass inherits from NSObject
@interface MyClass : NSObject
{
  int someValue;
  BOOL someFlag;
  NSString * theString;
}

+ (void) initializeSomething;
- (int) doSomething:(id)sender;

@end
```

Class Implementation

```
#import “MyClass.h”

@implementation MyClass
- (int) doSomething:(id)sender
{
  someFlag=YES;
  someValue=[self privateMethod]+[super doSomething:sender];
}

//methods don’t have to be declared in the header!
- (int) privateMethod { }
@end
```

More on ObjC

Categories

- A category let you add methods to an existing class
- Instance variables can not be added
- At runtime, the methods are just as much a part of the class as any other method defined on the class is

More on ObjC

Protocols

- Protocols declare methods not associated with a class, but which a class, or classes, can choose to implement to *conform* to the protocol
- If you claim to conform to a protocol, and don’t, it’s a compile time error
- Protocols effectively group objects by functionality, not class

More on ObjC

“Informal Protocols”

- They are categories, really
- No compiler error
- Help group functionality

Foundation Framework

Generally the "non-GUI" portion of Cocoa

- NSObject
- Memory Management
- Focus: NSString
- Focus: NSArray
- Focus: NSTask

NSObject

Root of the Cocoa Class Hierarchy

- Provides essential infrastructure methods like:

```
+ (id)alloc, - (id)init
- (BOOL) respondsToSelector:(SEL)aSelector
-  (Class)class
+ (NSString *)description
```

Memory Management

In a nutshell

- Cocoa uses reference counting
- **retain** adds a reference, **release** decrements it, and **autorelease** decrements it “later”
- The rules are simple:
    - Only creation methods (**alloc, copy, new**) return retained objects
    - **retain** to keep an object around
    - **release** when you are done with it
    - When there are no references, the object is deallocated

Memory Management

Example

- Managing the retain count

```
// We call alloc, so myString is implicitly retained
myString = [[NSString alloc] initWithFormat:@“%d”,100];

NSLog(myString);

// deallocate the memory for myString
[myString release];
```

Foundation Framework

Focus: NSString

- Opaque string class for use instead of char*
- “toll-free” bridged to CFStringRef
- Provides string manipulation methods and full Unicode support
- Use the @ “ . . ” construct to refer to a constant string in-code
- Use NSMutableString if you are changing a string as you go

Foundation Framework

Focus: NSString—Examples

```
NSString *someString = @“World!”;

NSLog(@“Hello %@”,someString);

NSLog(@“The length of the string is %d”,
      [someString length]);
```

Foundation Framework

Focus: NSArray

- Provides optimized “expanding” array support
- Takes NSObject subclasses only—but multiple types are allowed in one array
- Use with **NSEnumerator** or **objectAtIndex:**
- NSArrays retain their elements (and release on element removal), so you do not have to
- Use NSMutableArray if you are changing it as you go

Foundation Framework

Focus: NSTask

- Launch other processes
- Interact through NSPipes with stdin, stdout, stderr
- Wrap a command-line tool or shell script in a Cocoa GUI

Foundation Framework

Focus: NSTask—Example

```
NSPipe *outPipe;
NSTask *task = [[NSTask alloc] init];

[task setLaunchPath:@”/bin/ls”];
[task setArguments:[NSArray arrayWithObjects:@”-la”,
        @“/usr/bin”,nil]];

outPipe = [[NSPipe alloc] init];
[task setStandardOutput:outPipe];
[task launch];
```

Foundation Framework

Focus: NSTask—Example

```
NSPipe *outPipe;
NSTask *task = [[NSTask alloc] init];

[task setLaunchPath:@”/bin/ls”];
[task setArguments:[NSArray arrayWithObjects:@”-la”,
        @“/usr/bin”,nil]];

outPipe = [[NSPipe alloc] init];
[task setStandardOutput:outPipe];
[task launch];
```

Foundation Framework

Focus: NSTask—Example

```
NSPipe *outPipe;
NSTask *task = [[NSTask alloc] init];

[task setLaunchPath:@”/bin/ls”];
[task setArguments:[NSArray arrayWithObjects:@”-la”,
        @“/usr/bin”,nil]];

outPipe = [[NSPipe alloc] init];
[task setStandardOutput:outPipe];
[task launch];
```

AppKit

The parts of your app you can see

- Model-View-Controller (MVC)
- Provides application-level services
- Powerful and sophisticated text and drawing systems
- Infrastructure for events
- Controls

AppKit

Organizing your application: Model-View-Controller

- A standard design pattern leveraged by Cocoa

  The *Model* is your backend data

  The *View* is your AppKit-based front end

  The *Controller* is your custom class that ties the two together

- Enhances code factorization, encapsulation, and reuse

AppKit

Application-level services

- Handled by NSApplication via the global **NSApp** object
- **NSApp** does a lot of work for you
- Message **NSApp** to do application-level tasks

AppKit

Application-level services example

```
if ([NSApp isHidden])
        [NSApp setApplicationIconImage:anImage];
else
        [NSApp hideOtherApplications:self];
```

AppKit

Text System

- A complete, international, rich text editing solution in a drag-and-drop widget
- Automatic support for fonts, images, colors, cut/copy/paste, spell checking, printing, rulers, and more is built right in!
- NSTextView is the front end for a bunch of interlocking text classes
- A lot of power, complexity, and customization is there, but only if you need it

AppKit

Drawing

- NSBezierPath—**lineToPoint**, **curveToPoint**
- NSImage—**drawAtPoint**
- NSString, NSAttributedString—**drawAtPoint**
- NSColor, NSFont—**set**
- NSAffineTransform—**set**, **concat**
- NSGraphicsContext—**graphicsPort**
- CoreGraphics (Quartz) routines can also be used

AppKit

Event System

- NSApplication handles events
- Events get funneled to the focused NSResponder subclass—the “first responder”
- The first responder can/will automatically change

AppKit

Event System—Part 2

- If the first responder doesn’t respond to an event, the event is sent up the “responder chain” until an object is found that responds
- You can send actions to the first responder via “target/action” and targeting the first responder
- The dynamic nature of this system allows the responder chain to reconfigure itself on the fly

AppKit

Controls

- Controls typically have an *action*—a message that will be sent when the control is triggered
- Controls also have a *target*—the recipient object of the message
- NSButton, and NSMenuItem for example

AppKit

Back to the first responder

- The target can be determined on the fly!
- Cut/Copy/Paste work this way
- Menu items autoenable/disable

AppKit

Focus: NSTableView

- Similar to Carbon's DataBrowser control
- NSTableView does not hold its own data—it asks its data source object for cell content
- NSTableView handles all the drawing
- Data source objects should adopt the NSTableDataSource informal protocol

AppKit

Focus: NSTableView—NSTableDataSource Methods

- Two methods for a display-only table

```
-(int)numberOfRowsInTableView:(NSTableView *)tableView

-(id)tableView:(NSTableView *)tableView
    objectValueForTableColumn:(NSTableColumn *)tableColumn
    row:(int)row;
```

AppKit

Focus: NSTableView—NSTableDataSource Methods

- One more method to make the table editable

```
-(void)tableView:(NSTableView *)tableView
    setObjectValue:(id)object
    forTableColumn:(NSTableColumn *)tableColumn
    row:(int)row;
```

Demo

MP3 Player

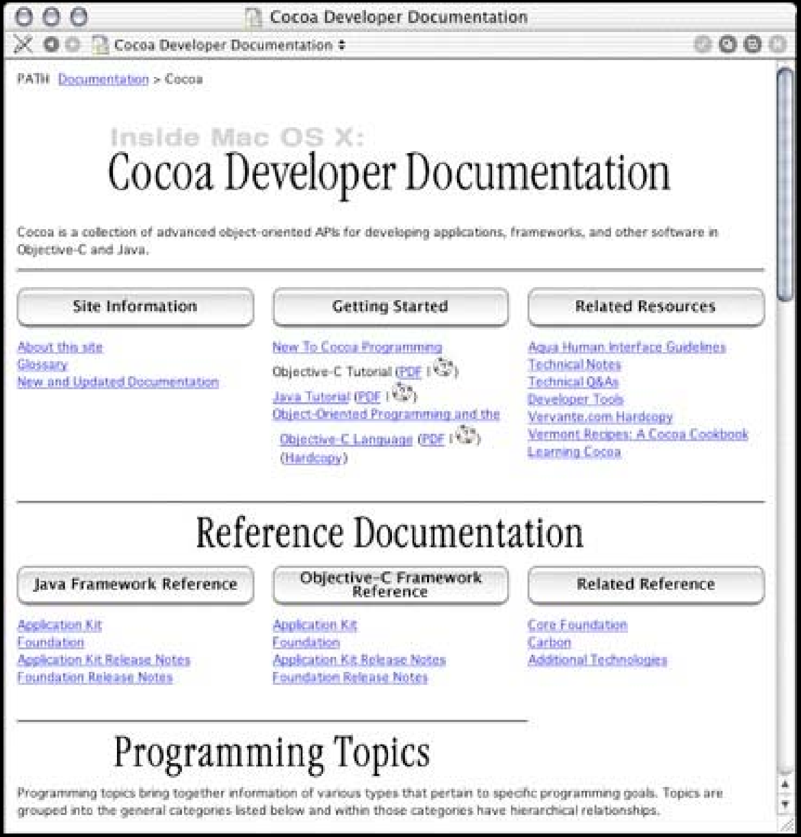
Matthew Formica
Cocoa DTS Engineer

Documentation

Matt Rollefson
Technical Publications

Documentation

Cocoa



Cocoa Developer Documentation

PATH Documentation > Cocoa

Inside Mac OS X:

Cocoa Developer Documentation

Cocoa is a collection of advanced object-oriented APIs for developing applications, frameworks, and other software in Objective-C and Java.

Site Information

About this site
Glossary
New and Updated Documentation

Getting Started

New To Cocoa Programming
Objective-C Tutorial (PDF |)
Java Tutorial (PDF |)
Object-Oriented Programming and the Objective-C Language (PDF |) (Hardcopy)

Related Resources

Aqua Human Interface Guidelines
Technical Notes
Technical Q&As
Developer Tools
Vervante.com Hardcopy
Vermont Recipes: A Cocoa Cookbook
Learning Cocoa

Reference Documentation

Java Framework Reference

Application Kit
Foundation
Application Kit Release Notes
Foundation Release Notes

Objective-C Framework Reference

Application Kit
Foundation
Application Kit Release Notes
Foundation Release Notes

Related Reference

Core Foundation
Carbon
Additional Technologies

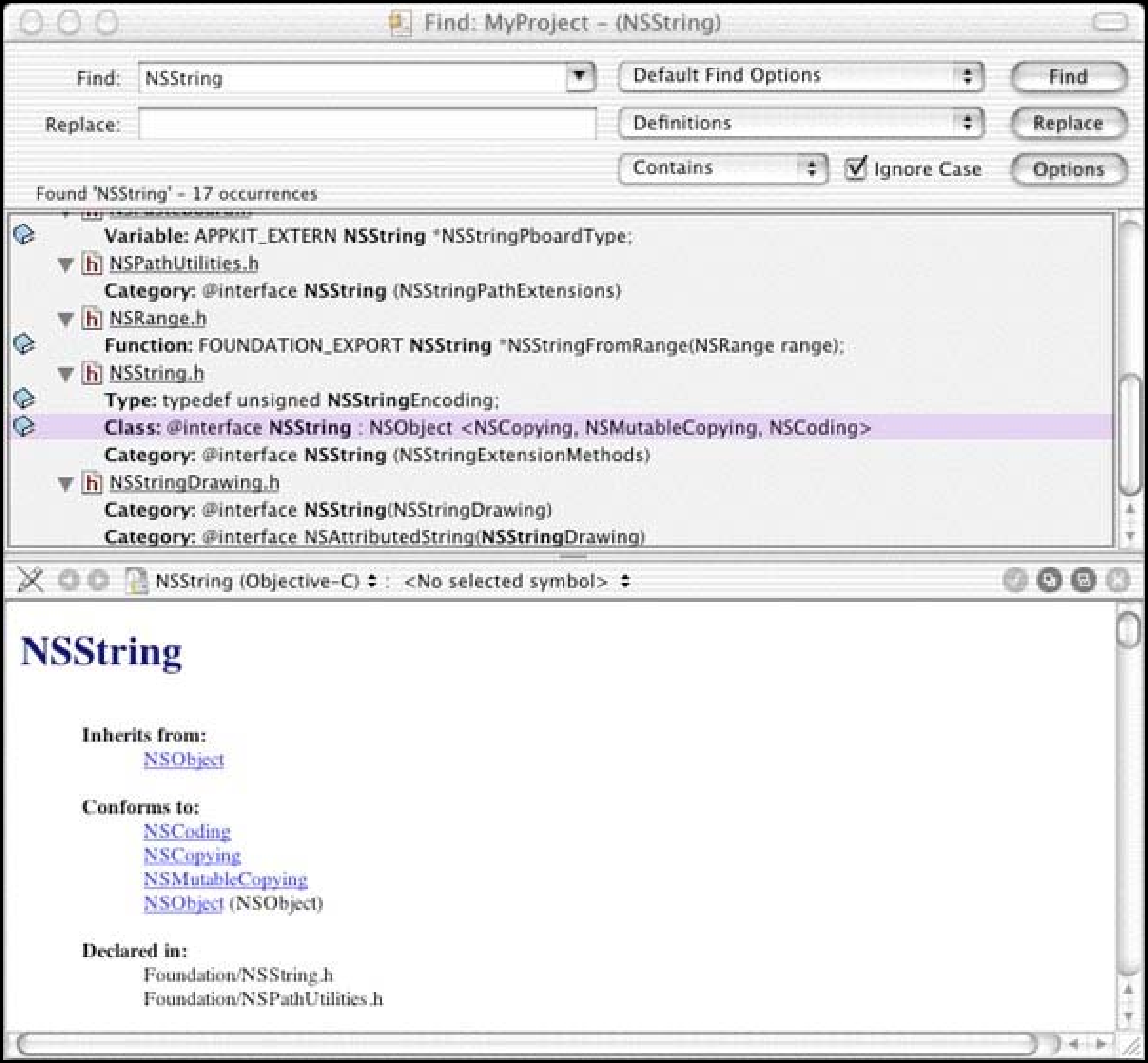
Programming Topics

Programming topics bring together information of various types that pertain to specific programming goals. Topics are grouped into the general categories listed below and within those categories have hierarchical relationships.

- Lots of content
- Reference API complete for 10.1
- More conceptual material on the way

Documentation > Cocoa
developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html

Documentation Access



Found 'NSString' - 17 occurrences

Variable: APPKIT_EXTERN **NSString** *NSStringPboardType;
NSPathUtilities.h
Category: @interface **NSString** (NSStringPathExtensions)
NSRange.h
Function: FOUNDATION_EXPORT **NSString** *NSStringFromRange(NSRange range);
NSString.h
Type: typedef unsigned **NSString**Encoding;
Class: @interface **NSString** : NSObject <NSCopying, NSMutableCopying, NSCoding>
Category: @interface **NSString** (NSStringExtensionMethods)
NSStringDrawing.h
Category: @interface **NSString**(NSStringDrawing)
Category: @interface NSAttributedString(**NSString**Drawing)

NSString (Objective-C) : <No selected symbol>

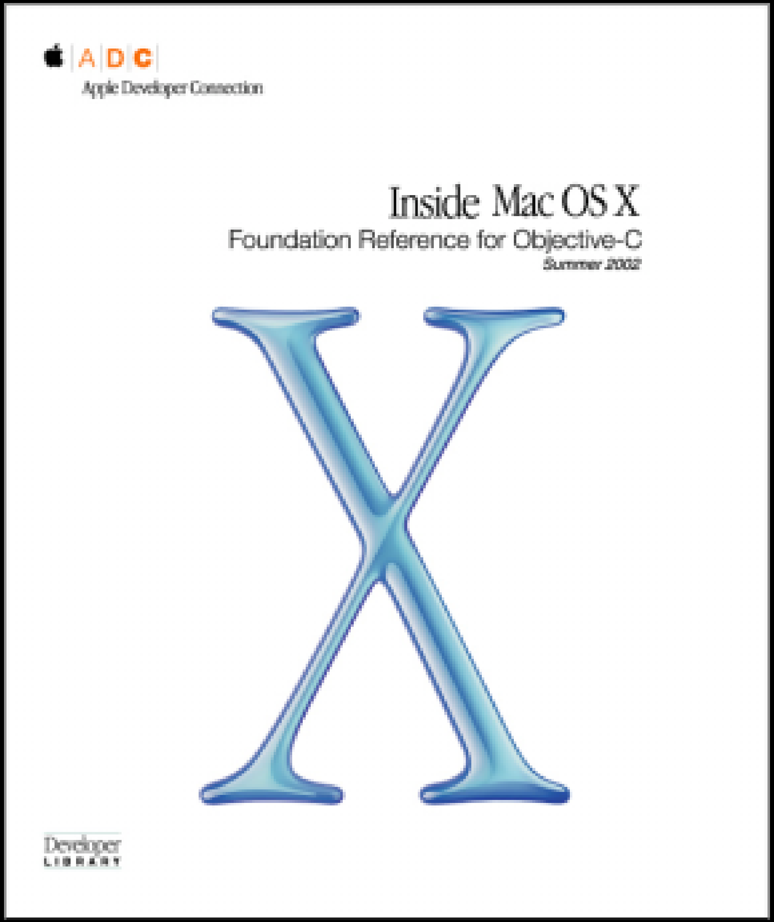
NSString

Inherits from:
NSObject

Conforms to:
NSCoding
NSCopying
NSMutableCopying
NSObject (NSObject)

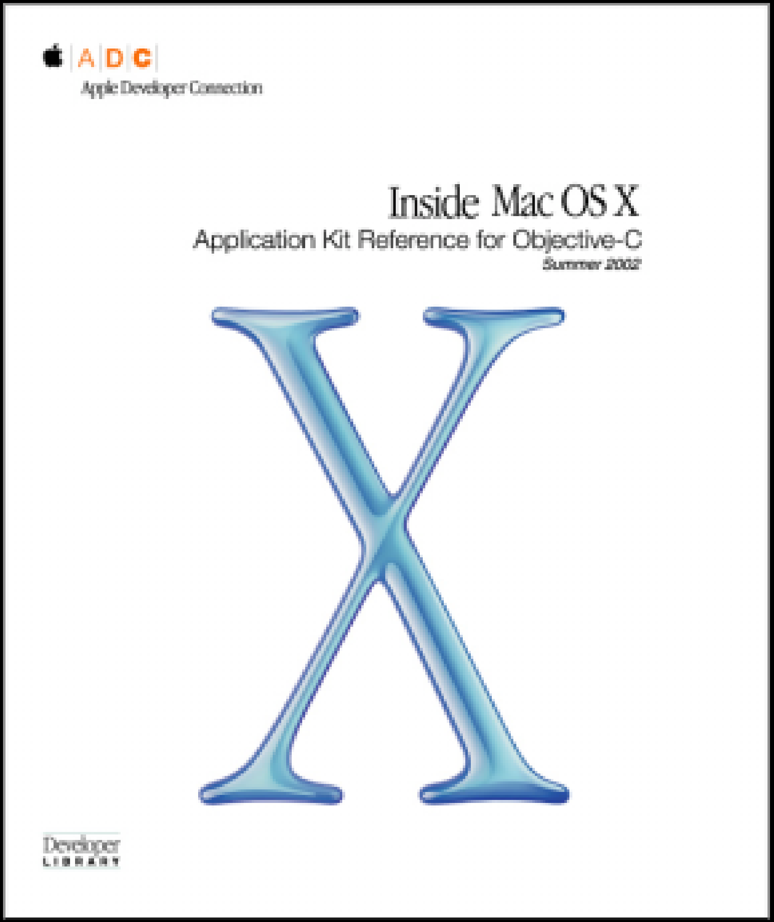
Declared in:
Foundation/NSString.h
Foundation/NSPathUtilities.h

Announcing . . .


ADC
Apple Developer Connection

Inside Mac OS X
Foundation Reference for Objective-C
Summer 2002

Developer
LIBRARY


ADC
Apple Developer Connection

Inside Mac OS X
Application Kit Reference for Objective-C
Summer 2002

Developer
LIBRARY

Roadmap

301 Cocoa: What's New	Civic **Tues., 9:00am**
302 Cocoa API Techniques: Understanding, leveraging, and extending	Hall 2 **Thurs., 9:00am**
303 Cocoa Scripting: Scripting overview and recent changes	Room A2 **Thurs., 10:30am**
304 Cocoa Controls and Accessibility: Overview of controls; new Accessibility APIs	Room A2 **Thurs., 5:00pm**

Roadmap (Cont.)

305 Cocoa Drawing: 2D graphics in Cocoa: Images, bezier paths, . . .	Hall 2 **Fri., 10:30am**
306 Cocoa Text: In-depth overview of the text system	Room J **Fri., 2:00pm**
FF016 Cocoa Feedback Forum: Comments and suggestions for Cocoa	Room A1 **Fri., 5:00pm**

Who to Contact

Heather Hickman
Cocoa Evangelist
hhickman@apple.com

For More Information

- O’Reilly “Learning Cocoa” and “Building Cocoa Applications: A Step-by-Step Guide”
- Cocoa Developer Documentation
  http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html
- Apple Customer Training
  http://train.apple.com/

WWDC2002

WWDC2002

WWDC2002